A Composition Algorithm for the SILO Cross-Layer Optimization Service Architecture

Manoj Vellala, Anjing Wang, George Rouskas, Rudra Dutta
NCSU, Raleigh, NC
{mvellal, awang, rouskas, rdutta} @ncsu.edu

Ilia Baldine
RENCI, Chapel Hill, NC
ibaldin@renci.org

Daniel Stevenson
RTI, RTP, NC
dstevenson@rti.org

***Abstract*— We have previously proposed a software architecture for the future Internet called SILO that is specifically targeted to accommodate cross-layering gracefully. In such an architecture, composing the services that make up the software layers for specific data flow requirements emerges as an essential part of the architectural system. We provide a minimal set of precedence constraints to express service interactions, and an algorithm that obtains correct compositions under this set. The algorithm appears to have a super-polynomial worst case running time, and we conjecture that the problem is NP-complete. We show that under some further simplifications of the set of constraints, the problem is polynomial.**

I. Introduction

One of the new frontiers of networking is pervasive and ubiquitous networking, enabled by wireless, mobile devices with low power budgets and other limitations not previously considered common for data networks. Such networks often have unique requirements of *cross-layer interactions and optimizations*; e.g. power-aware routing [1]. Current internetworking protocols are flexible enough to handle such solutions, but require custom solutions which make combination of technologies and evolution difficult if not impossible. In some cases, the overhead imposed in bending the capability of such devices to existing network architecture can make the use of such devices prohibitive and even useless.

Currently, a consensus appears to be forming within the community regarding the need to think carefully about the requirements for the Internet in 15-20 years, and to carry out a focused research agenda to realize this vision, possibly starting with a "clean slate". A new initiative of the National Science Foundation has targeted this issue [2]. Our project team is currently working on a project that is part of this initiative [3], with the ability to integrate cross-layer design solutions as a primary goal. We have previously described the general characteristics of our SILO architecture (Services Integration, controL, and Optimization) elsewhere [4], [5]. We briefly describe the essential points in the next section, and go on to define the problem of composition.

II. The Composition Problem

Fundamentally, the SILO architecture generalizes the concept of layering. The building block is a service, which takes the place of a protocol layer. Like a protocol layer, it presents a data interface to a served (upper) and serving (lower) service (layer), but in addition, it provides (i) a control interface, which communicates with a unified control agent, and (ii) a set of rules for composability. This opens the door to creating unique protocol stacks with custom cross-layer interaction for distinct class of applications, or even per individual data flow. Thus there is no uniform protocol stack that can/need be assumed to be present on every node. Instead, protocol stacks must be composed on demand out of available services. We assume that there is a global ontology of services that is available to end nodes, perhaps offline (like the "ports" file on UNIX). However, different services themselves may not be available at differnt nodes, e.g. a small sensor node may not have a sophisticated transport service.

A service S_1 may require that another service S_2 be present in the protocol stack, and somewhere closer to physical channel; this is an example of a precedence constraint imposed by S_1. For example, in a topology-controlled wireless network, an application or transport layer can utilize geographic addressing, and depend on a lower layer service to convert to a specific address depending on the current power-controlled topology; this means that some of the application or transport logic must be embedded below the layer that is aware of the power-controlled topology. To express this, a relation *RequiresBelow* must be defined in the ontology of services. Alternatively, two relations *Requires* and *MustOccurAbove* can be defined and used with an AND condition, to same effect. Similarly one can conceive of relations such as *MustOccurBelow* or *Forbids*. We allow only first-order relationships, i.e. we do not provide any way to make assertions equivalent to "if S_1 and S_2 both occur in the stack, then S_1 must be above **if** S_3 is also present, but otherwise must be below S_2". However, the first-order relationships can be combined using AND or OR connectives.

We propose the following minimal set of precedence constraints:

Requires,
MustOccurAbove (*Above*),
MustOccurImmediatelyAbove (*ImmAbove*),
MustNotOccurImmediatelyAbove (*NotImmA*).

The latter three are interpreted as "if S_1 and S_2 both occur ..." conditions. We assert that this set of constraints allows all first-order relationships to be expressed. We omit a formal proof; it consists of listing every possible combination of two services in a stack, and showing that any subset of them can be allowed (and the rest disallowed) by combining these. For example, *Forbids* can be obtained by specifying both S_1 *MustOccurAbove* S_2 and S_2 *MustOccurAbove* S_1. ontology.

[1]This research was supported in part by NSF grant # Nets-FIND:0626103.

III. Composition Algorithm

The composition problem then can be stated as follows: given a set of services in the ontology, to obtain an ordering that is consistent with the precedence constraints in the ontology, possibly augmenting the set of services for the purpose. A straightforward approach can easily guarantee that correct stacks (obeying all constraints) are constructed. Briefly, the steps are:

Initialize *essential* services list from application specification
Designate any service S_1 as the top service
  Recursively, build the stack below
    S_i := service last added
    If S_i *Requires* any service, add to *essential*
    If S_i has an *ImmAbove* constraint, add it to stack
      (unless marked backtrack)
    Otherwise, add any other service which can be added
  Recurse
  // (No other service can be added)
  Check to see if stack violates any *Above* condition
    or any essential service is missing
  If not, output stack and exit
  Else backtrack to remove last service added
Start with some other service as top service

Clearly, this algorithm will produce correct stacks; equally clearly, it has a very long running time. Many improvements to running time can be made in the form of sophistications such as checking each service for violations with services already on the stack, backtracking several steps when removing a service required by another above, starting with essential services as top service choice, etc. However, they leave the algorithm essentially the same, and not guaranteed to complete in polynomial time. For this reason as well as some indications from a graph-theoretic modeling, we conjecture that the problem is NP-complete; however, we do not have a formal proof at this time. In practice, in ontologies with reasonable sets of constraints, the version of the algorithm with all the accelerations runs with very little delay (a few seconds with an ontology of around fifty services) if there are a reasonable number of constraints to prune the search.

However, this changes with further drastic simplification of the set of precedence constraints. If there are no essential services designated by the application, and only one of the three ordering constraints are allowed, together with *Requires*, the problem can be solved polynomially. For ease of discussion, w.l.o.g. we consider a unique top service S_s and a unique bottom service S_e. Further, we consider that the set of services A_s that can follow S_s are known, as is the set of services B_s that can precede S_e. In the general case, these can of course be the set of all services. We consider a digraph where every service can be represented as a node and the constraints as edges between them.

a) Above: Between all services excluding S_s and S_e, the edges are of type *MustOccurAbove* if an edge exists. If no edge exists between two nodes then there is no ordering restriction between them. The problem is finding a directed ordering from S_s to S_e. If one of A_s is also one of B_s, then S_s, A_s, S_e is a valid ordering and this take linear time in number of services to check. For the case that an edge exists from a B_s to an A_s, if there exists any vertex v such that, $v \rightarrow A_s$ or $B_s \rightarrow v$ or both are not true, then A_s, v, B_s is a valid ordering. If no such v exists then there is no ordering satisfying the constraints. Finding such a v is again linear in number of services for a given pair of A_s and B_s. In the first two cases the ordering is of length 4 at maximum and in the second case it is of length 5. Each of the sub-cases is of polynomial time complexity.

b) ImmAbove: The proof is exactly the same as for the case above. An easy way to see this is as follows: if between two services A_s, B_s there are no edges then we can include the edges $A_s \rightarrow B_s$ and $B_s \rightarrow A_s$ and then run Dijkstra's algorithm. Here the services S_s and S_e are treated like any other service.

c) NotImmA: Here we construct another graph in which every edge is an *ImmAbove* edge. Initially this is a complete bi-directed graph on the nodes (services). Then remove the directed edges between any two services u, v if a constraint u *NotImmA* v is specified. Now run Dijkstra's algorithm.

Obviously, even when *ImmAbove* and *NotImmA* are both present, the problem can be solved in polynomial time by removing the (directed) edges corresponding to the contradicting constraints. (Such contradicting constraints should not exist if the ontology is consistent). Finally, we note that in the more general case when *ImmAbove* and *Above* are both present, the problem can *still* be solved in polynomial time by the same procedure as in the *Above* only case. However, the most general case remains beyond our reach.

IV. Conclusion

In this paper, we have articulated the problem of composing a given partial set of services into a protocol stack obeying pre-specified precedence constraints in a framework of composable protocols. We have provided a minimal sufficient set of such constraints, and a correct composition algorithm. We conjecture that this problem is NP-complete, though it is polynomial under simpler conditions. Settling the complexity of the problem under the minimal set, as well as designing more efficient composition algorithms, is part of our ongoing work. Our ongoing ontology, as well as service and application APIs, is available at the project website [3].

References

[1] V. Srivastava and M. Motani, "Cross-layer design: A survey and the road ahead," *IEEE Communications Magazine*, vol. 43, no. 12, pp. 112–119, December 2005.

[2] National Science Foundation, "Future internet design website," http://www.nets-find.net/.

[3] "Services integration, control, and optimization," http://www.net-silos.net/.

[4] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The silo architecture for services integration, control, and optimization for the future internet," in *Proceedings of IEEE ICC, Glasgow, Scotland*, June, 2007.

[5] I. Baldine, M. Vellala, A, Wang, G. N. Rouskas, R. Dutta, D. Stevenson, "A Unified Software Architecture to Enable Cross-Layer Design in the Future Internet," in *Proceedings of IEEE ICCCN, Turtle Bay, Hawaii*, August, 2007, pp. 26-32.